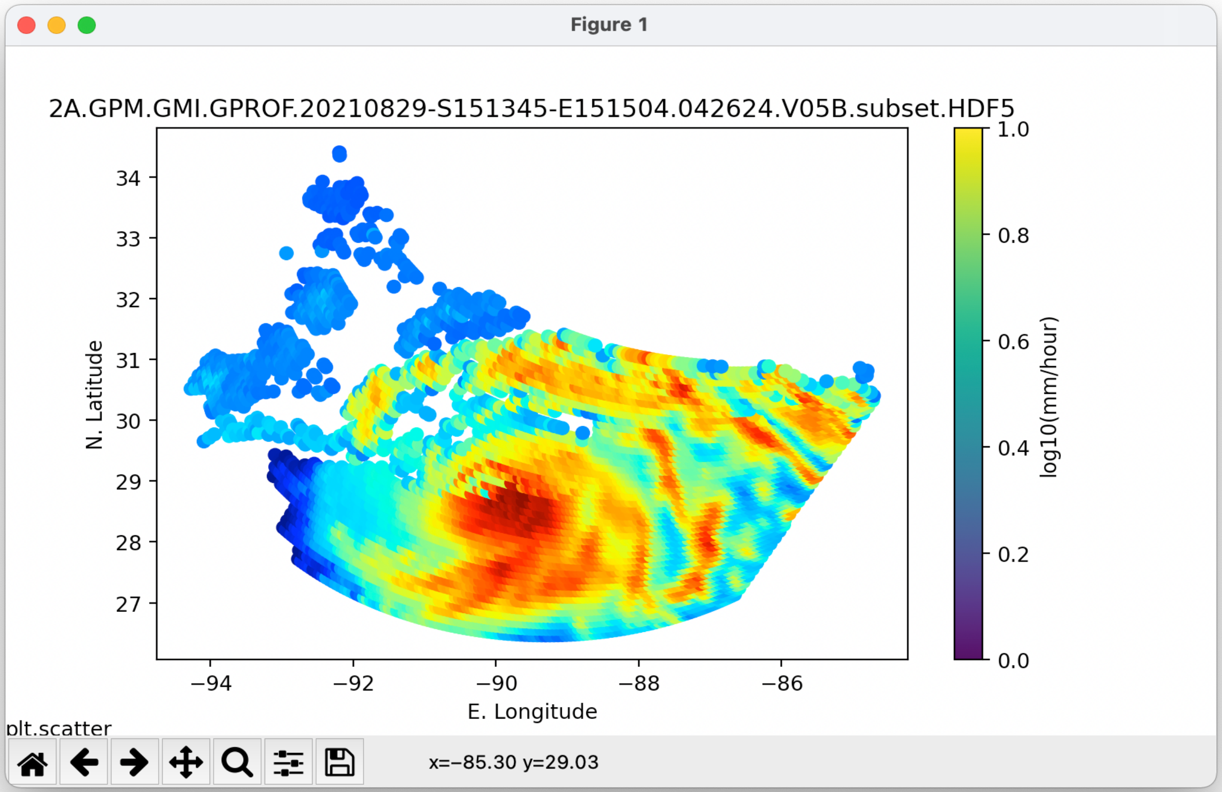
Using Python to Read Data Products from the Global Precipitation Measurement (GPM) Mission

The Precipitation Processing System at NASA Goddard
https://arthurhou.pps.eosdis.nasa.gov/

Updated: 25 January 2022
by Owen Kelley (Owen.Kelley@nasa.gov)



Caption: A Python visualization of data from an HDF5 file containing precipitation retrievals calculated from observations made by the GPM Microwave Imager (GMI). The image shows an overflight of Hurricane Irene in the Gulf of Mexico on 29 August 2021 at 1514 UTC, at the time that the hurricane was about to make landfall in Louisiana. Because rainfall rates are typically logarithmically distributed, the colors are defined by the log base 10 of rainfall rate (i.e., 0 and 1 in the color bar correspond to rainfall rates of 1 and 10 mm h^{-1}, respectively).

Contents

1. Introduction

NASA's Global Precipitation Measurement (GPM) core satellite carries a radar and a passive microwave instrument and has been in Earth orbit since February 2014 (Hou et al. 2014). GPM standard data products are written the HDF5 file format, although some GPM-derived products are written in the NetCDF4 or GeoTIFF format. All of these formats can be read with Python, a high-level programming language that is popular among scientists. The present document gives tips on how to use Python to read GPM data, the time the observations were made, and the data's geographic location. This document is intended for people who are familiar with Python but not necessarily HDF5 or GPM.

GPM data may possess one of two geometries: swath or grid. A swath file contains the data collected by a satellite instrument as the satellite orbits once around the Earth. In a swath file, the latitude and longitude are stated explicitly for each of the instrument's fields of view. The latitude and longitude stated is the field of view's center point on the Earth ellipsoid. By convention, GPM data products state location on the Earth ellipsoid even if the instrument collects information along a slant path through the atmosphere. A grid file contains arrays with at least two dimensions, one of which represents latitude and another of which represents longitude. In contrast to a swath file, a grid file contains averages or accumulations of values from multiple swaths, usually over a month-long period. Swath and grid files are discussed in separate sections in the present document. The sample Python program also contains separate functions for reading swath and grid files.

The most widely used GPM data product is a gridded product called IMERG, which stands for "Integrated Multi-satellitE Retrievals for GPM." IMERG combines data from multiple satellites and other data sources to create a global estimate of surface precipitation rate in millimeters per hour. The IMERG algorithm outputs HDF5 files covering either 30 minutes or one month, with the record extending from June 2000 to the present. IMERG's 1800-by-3600-element grid covers the globe at 0.1º×0.1º resolution. The highest-quality estimates are calculated several months after the satellite observations are made (Final IMERG). In near real time, a simplified version of the algorithm runs using the data that are available that quickly (Early IMERG and Late IMERG) (Huffman et al. 2019).

While the direct output of the IMERG algorithm is written in the HDF5 format, there are IMERG-derived products written in the NetCDF4 or GeoTIFF formats. For many years, HDF5 was one of the formats that NASA recommended for storing Earth-observing data, but recent high-level guidance from NASA suggests that NetCDF4 and GeoTIFF are now preferred (DPDG Working Group 2020). The Goddard DAAC (i.e., the Goddard Earth Sciences Data and Information Services Center, GES DISC) creates an IMERG-derived product stored in the NetCDF4 format. These NetCDF4 files contain a 24-hour average of the standard 30-minute precipitation estimates generated by the IMERG algorithm. The Precipitation Processing System (PPS) at NASA Goddard produces an IMERG-derived data product that is stored in the GeoTIFF format. These GeoTIFF files contain 30-minute, 3-hour, 24-hour, 3-day, 7-day, or 1-month averages of the estimates generated by the IMERG algorithm.

While the present document focuses exclusively on Python, Python is not necessarily the best language in all situations. Three general options for reading GPM files include a low-level language (C or FORTRAN), a high-level language (IDL, Matlab, or Python), or a Geographic Information System (GIS) such as QGIS or ESRI ArcGIS (https://qgis.org; https://esri.com). The advantage of a high-level language is that fewer lines of code need to be written, compared to a low-level language. The rest is a matter of opinion (Lutz 2013, pp. 21–22). Python is a high-level language with the advantage that it is free to use. High-level languages for which you purchase a license (e.g., IDL and Matlab) have a reputation for being stable, being easy to install, and providing tech support. Between 2008 and 2020, Python had two somewhat-incompatible versions concurrently in widespread use, Python 2 and Python 3 (Lutz 2013). Only Python 3 is supported now, but there is concern that significant changes in each major release of Python 3 might break Python libraries that many researchers rely on (Lutz 2020).

The numpy, matplotlib, h5py, NetCDF4, PIL, and datetime libraries must be installed in your copy of Python for some functions in the sample Python program to read GPM data products as described in the present document. One way to install Python 3 and some or all of the needed libraries is to install the free Anaconda distribution of Python 3. For Linux or Mac users, determine if you have Python 3 installed and in your path by typing `python` on the command line. For Microsoft Windows users, type `python` or `anaconda` in the search field in or near the Start menu.

A few disclaimers are worth mentioning. The Python libraries used in this tutorial might not run in future versions of Python, and it is beyond the control of the author if that occurs. The Python libraries used in this tutorial are not necessarily the best or most stable ones. They are, however, adequate for illustrating the logical structure of the GPM data, and they at least suggest the steps that are needed to read and interpret GPM data products. NASA reprocesses GPM data every few years with improved science algorithms, and in a future reprocessing, the names of variables in the data products may change. It is possible that this document will state a different variable name than in the version of GPM data products that you are examining.

2. Download Data and Sample Program

The code examples in the present document are based on a Python program that can be downloaded from the PPS website. To obtain this Python program and the sample data files that it reads, visit https://gpmweb2https.pps.eosdis.nasa.gov/pub/THOR/python and download the file called `gpmPython.zip`. On a Linux or Mac system, type `unzip gpmPython.zip` to extract the files from the *.zip file. On a Windows system, right click on the *.zip file and select "extract files." Below are the contents of the `gpmPython.zip` file:

```
2A.GPM.GMI.GPROF.20210829-S151345-E151504.042624.V05B.subset.HDF5
2A.GPM.Ku.V9.20211203-S003806-E004322.044108.V07A.subset.HDF5
3B-DAY-GIS.MS.MRG.3IMERG.20210519-S000000-E235959.4140.V06B.tfw
3B-DAY-GIS.MS.MRG.3IMERG.20210519-S000000-E235959.4140.V06B.tif
3B-DAY.MS.MRG.3IMERG.20210519-S000000-E235959.V06.nc4
3B-HHR.MS.MRG.3IMERG.20210519-S000000-E002959.0000.V06B.HDF5
gpm.py
gpmPythonNotes.pdf
```

The first two of the data files contain single-instrument rainfall estimates from the main GPM instruments: the Dual-frequency Precipitation Radar (DPR) and the passive microwave radiometer called the GPM Microwave Imager (GMI) (Hou et al. 2014). To keep the *.zip file from becoming too large, these DPR and GMI files are subsets of the original full-orbit HDF5 files that are stored in the GPM online archive. These subsets are geographically smaller (they contain less than a full orbit of data), and they contain only a sample of the variables in the archived files. These two subsets were generated using the subset capability of the PPS data ordering system that is called STORM (http://storm.pps.eosdis.nasa.gov).

In addition, the `gpmPython.zip` file contains three formats of the IMERG multi-satellite precipitation data product. These formats are a GeoTIFF file, a NetCDF4 file, and an HDF5 file.

Once you have finished running the program on the sample data files, you may wish to download other GPM data files from the online archive hosted by PPS. Before doing so, register your email address with PPS by going to this URL: http://registration.pps.eosdis.nasa.gov/. Once registered, visit https://jsimpsonhttps.pps.eosdis.nasa.gov/ to download near real-time GPM HDF5 files, or visit https://arthurhouhttps.pps.eosdis.nasa.gov/gpmdata/ to download research-quality GPM HDF5 files. These download sites can be accessed using a web browser or the Linux `wget` and `curl` commands. If using a web browser, type in your just-registered email address when prompted for a username and password. Alternatively, the same GPM HDF5 files can be downloaded from the online archive hosted by the Goddard Earth Sciences (GES) DISC: https://earthdata.nasa.gov/eosdis/daacs/gesdisc.

3. Reading Data from HDF5 Files

3.1. Reading HDF5 Files with the Sample Program

Start a Python 3 session. On a Linux system, type `python` on the command line. On a Microsoft Windows system, one may start a Python session using the Start menu.

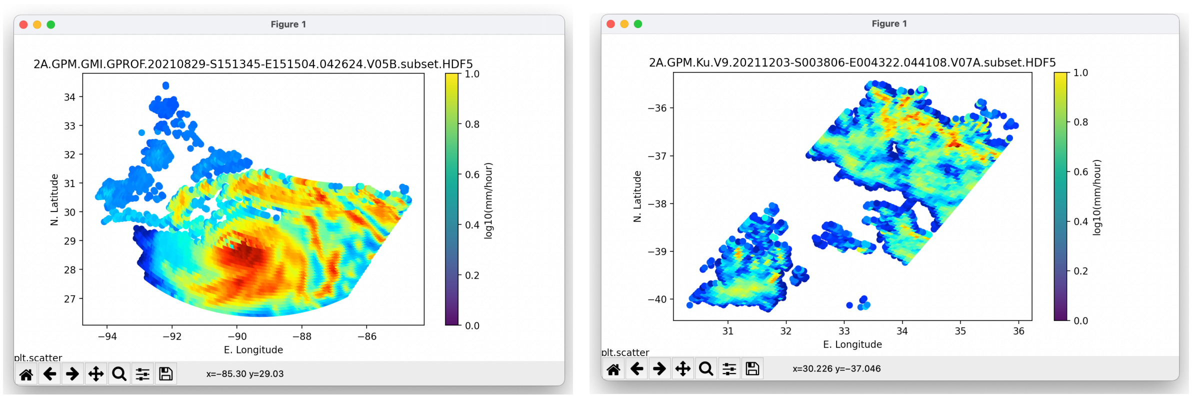
Load the `gpm.py` sample program by typing `import gpm` on the Python interactive prompt. This Python source file is one of the items in the `gpmPython.zip` that is available for download from the PPS website as described in the previous section. Once the `gpm` module is loaded with the `import` command, one can type `help(gpm)` to obtain information about the functions in that module.

Using a text editor, examine the contents of the `testRun()` function near the bottom of the `gpm.py` Python source file. One may pick commands from `testRun()` to run interactively, remembering to prefix their names with `gpm` and a period. For example, to run the `readSwath()` function, one would type `gpm.readSwath(fileName,varName,swath)` on the Python interactive command line. In preparation, one would define the two input variables, `fileName` and `varName`, and initialize the `swath` output variable as an empty dictionary using the command `swath={}`. Alternatively, run the entire suite of tests by typing `gpm.testRun()` on the interactive Python command line. When executing `gpm.testRun()`, Python plot windows will pop up. Such a window should be dismissed in order to advance to the next test.

Python has several functions for examining objects. The `help()` and `type()` functions work on variables, functions, and other objects. To list the methods and attributes of an object, some Python environments will allow you to type the name of that object on the interactive Python command line followed by a period and then hit the tab key twice rapidly. This feature is similar to the command-line completion feature provided by many Linux shells.

There are many ways to modify a Python source file, including integrated development environments (IDEs), source-code editors, and simple text editors. The simplest solution would be to use the `vi` text editor under Linux and MacOS or to use a text editor like WordPad under Microsoft Windows. After editing the source file, load the modified code by typing `importlib.reload(gpm)` on the Python interactive command line. In Python 3, the `reload()` function becomes available only after executing `import importlib`.

The `gpm.readSwath()` function in the `gpm.py` program reads data and the associated latitude and longitude, returning them in a Python dictionary called `swath`. The `swath` dictionary can then be passed to the `gpm.plotSwath()` function. The images generated from the GMI and DPR swath files are shown in Figure 1.



(a) **(b)**

Fig. 1. Python display of GPM single-instrument HDF5 files created by the `gpm.plotSwath( )` function in the `gpm.py` sample program. (a) GMI surface precipitation rate (millimeters per hour). (b) DPR surface precipitation rate (mm h^{-1}).

The grids in GPM data products either cover the whole globe or just 70°S to 70°N. The `gpm.readGrid()` function reads a specified grid variable and `gpm.plotGrid()` displays it graphically.

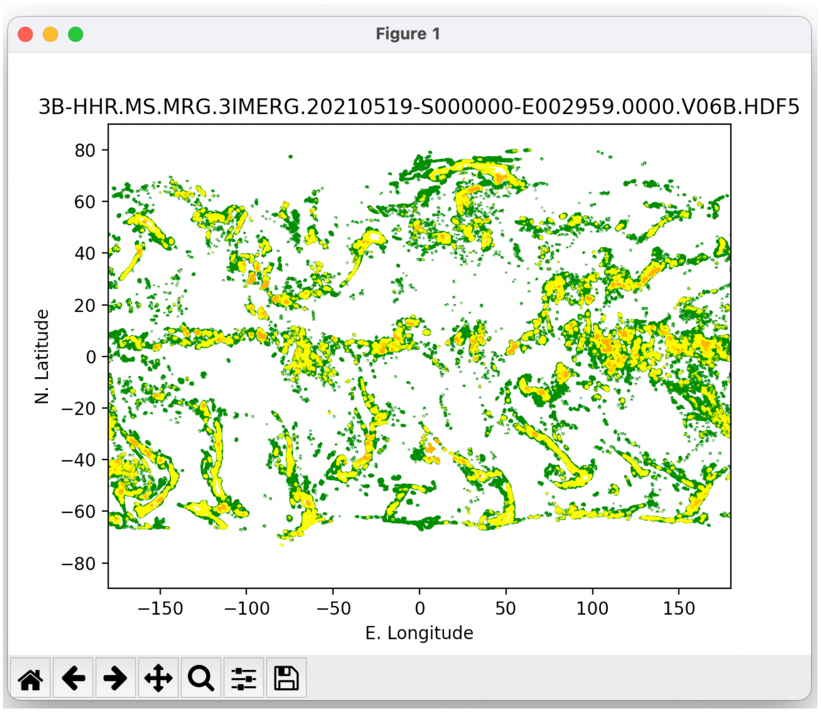


Fig. 2. Python display of the GPM multi-satellite product called IMERG, created by the `gpm.plotGrid()` function in the `gpm.py` sample program. The display is the base-10 logarithm of 30-minute averaged rainfall rate (mm h^{-1}).

The GPM file specification document names all of the variables in GPM HDF5 files. To access them in Python (or in IDL or Matlab for that matter), it can be helpful to know the absolute path to the variable. The GPM file specification is available as a PDF file from the PPS website, https://arthurhou.pps.eosdis.nasa.gov/. The absolute path is usually the name of a group or groups followed by the name of a variable, such as `/S1/Latitude`. To list the absolute path for all datasets in an HDF5 file, use `gpm.listContents()` from the sample `gpm.py` Python program. The one mandatory argument is the name of the HDF5 file to be examined. If only one argument is provided then the output is sent to the screen. If the name of an output text file is provided as an optional second argument, then the list of variables is printed to that file.

When editing the sample program, it may be helpful to insert "pauses." In other words, a place where the program states the current location in the source file, the program pauses execution, and the program returns control to the interactive Python command line. The program's local variables are available for examination. To insert such a pause, use the following

Python commands:

```
import code
import traceback
traceback.print_stack(limit=1)
code.interact(local=locals())
```

When one has finished the examination from the interactive prompt, one can resume the program's execution by typing the following command on the Python interactive command line: control-d on Linux and MacOS or control-z on Microsoft Windows. This sort of pause is analogous to the functionality of the "stop" command that can be inserted in source code of a program written in the Interactive Data Language (IDL) and the ".con" interactive IDL command.

3.2. Reading HDF5 Files with h5py

Once you know the full path to the HDF5 variable that you want to read, reading it in Python take just a few lines of code: import the h5py library, open the file, and read the variable. For example, to read the GMI surface precipitation estimate from a 2A GPROF GMI HDF5 file, one would store the file's name in the `fileName` variable and execute these lines of code:

```
import h5py
import numpy
fileHandle = h5py.File( fileName, 'r' )
data = numpy.array( fileHandle['/S1/surfacePrecipitation'] )
```

It is tempting to think of the objects returned by h5py's `fileHandle[varName]` as an array but actually it is an HDF5-specific object type (Collette 2013, Chap. 3). The `numpy.array()` function call converts the object into a numpy array, which is usually a good idea because numpy arrays "play nice" with many other Python functions used in data analysis. Some numpy functions will work on the object returned by h5py. For example, dimensions and other characteristics can be printed in the following way:

```
data.shape
import numpy
numpy.amin( data )
numpy.amax( data )
numpy.median( data )
numpy.mean( data )
```

However, if one types `type(data)` or just `data` on the interactive Python prompt, the output reveals that the object is not a numpy array if one is looking at the direct output of h5py's `fileHandle[varName]`:

```
<HDF5 dataset "surfacePrecipitation": shape (229, 221), type "<f4">
```

If one subsets an HDF5 dataset, the h5py function returns a pure numpy array. For example, a subset operation that takes every tenth element of an HDF5 dataset, i.e., `type(data[::10,::10])`, will return `<type 'numpy.ndarray'>`.

Another common operation that researchers perform on data read from GPM HDF5 files is to see what fraction of a variable exceeds a particular threshold. For example, one might want to determine what fraction of observations in an array were determined to have zero rain rates and what fraction had heavy rain of equaling or exceeding 10 millimeters an hour. The following two Python statements calculate such fractions.

```
print( 'fraction of elements equal to 0: ', \
  1.0* numpy.sum( numpy.equal(data, 0) ) / numpy.size(data) )
print( 'fraction of elements greater than or equal to 10: ', \
  1.0* numpy.sum( numpy.greater_equal(data, 10) ) / numpy.size(data) )
```

To make a quick plot of a numpy data array, the matplotlib library's pyplot module can be used. To get a meaningful plot, one does need to use care in the handling of the missing data values common in GPM variables. The missing data values are often large negative numbers, such as -9999.0. Use the following Python commands:

```
import matplotlib.pyplot as plt
import numpy
# -- replace missing data value
missing = numpy.amin( data )
missingInImage = -1
image = numpy.where( data==missing, missingInImage, data )
# -- display data
plt.imshow( image )
plt.show()
```

4. Reading Geolocation from HDF5 Files

4.1. Reading Geolocation from a Grid File

The grid boxes in a GPM grid file are regularly spaced in latitude and longitude. Such a grid can be plotted on a map given merely the ranges of latitude and longitude covered by the grid. In some GPM grid files, each grid box covers a 5º latitude by 5º longitude area, while in other GPM grid files, each grid box is 0.1º × 0.1º.

GPM grid arrays either cover the entire globe or cover a range of latitude, such as 70ºS to 70ºN. The latitude and longitude ranges are stated in the `GridHeader` annotation written to the HDF5 group that holds the variables of that grid. The grid folder is a top-level object in the HDF5 file. In an IMERG HDF5 file the grid folder is called `/Grid`. To read the `gridHeader` from an IMERG file, store the name of the HDF5 file in a string variable called `fileName`. Then execute the following Python commands:

```
gridName = '/Grid'
# -- read from file
with h5py.File( fileName, 'r' ) as fileHandle:
  GridHeader = fileHandle[ gridName ].attrs['GridHeader'].decode('utf-8')
# -- convert from one long string to a dictionary of parameters
GridHeader = GridHeader.split( ';\n' )[0:-1]
GridHeader = dict( nameValue.split('=') for nameValue in GridHeader )
```

The resulting `GridHeader` dictionary contains the following elements for an IMERG HDF5 file. In this printout, extra white space is added for readability.

```
{'BinMethod':                 'ARITHMETIC_MEAN',
 'Registration':              'CENTER',
 'LatitudeResolution':        '0.1',
 'LongitudeResolution':       '0.1',
 'NorthBoundingCoordinate': '90',
 'SouthBoundingCoordinate': '-90',
 'EastBoundingCoordinate':  '180',
 'WestBoundingCoordinate':  '-180',
 'Origin':                    'SOUTHWEST'
}
```

In addition to the information in the `GridHeader` annotation, the latitude and longitude ranges of GPM gridded datasets are stated in the GPM file specification document that is available on the PPS homepage, https://arthurhou.pps.eosdis.nasa.gov/. For the IMERG HDF5 half-hour and monthly files, and no other GPM data product, there is a third way to obtain the geographic information needed to plot the grid on a map. Specifically, IMERG HDF5 files contain one-dimensional `lat` and `lon` arrays that store the center of each grid box's latitude and longitude.

4.2. Reading Geolocation from a Swath File

For each array in a GPM swath file, there are variables called `Latitude` and `Longitude` associated with it. A GPM swath file may contain multiple variables called `Latitude` and `Longitude`. The appropriate `Latitude` and `Longitude` variables are either located in the same HDF group that contains a swath array or in the parent group of that HDF5 group. The `gpm.readSwath()` function of the `gpm.py` Python program checks both of these places for arrays called `Latitude` and `Longitude`. The `gpm.readSwath()` function stores these two arrays along with the data of interest in a Python dictionary named `swath`. The Python command to create the dictionary is as follows after the `lat`, `lon`, and `data` arrays have been created:

```
swath = { 'lat': lat, 'lon': lon, 'data': data }
```

One would access one of the elements of the `swath` dictionary, such as the latitude element, with the following syntax:

```
swath['lat']
```

5. Reading Observation Date and Time from HDF5 Files

5.1. Reading Datetime from a Grid File

While a GPM swath file has a separate datetime stamp for each scan in the file, a GPM grid file has a single datetime range that defines the period covered by the data in that file. A scan is a row of instrument fields of view that are oriented approximately perpendicular to the satellite's direction motion as it orbits around the Earth. All GPM HDF5 files, whether swath or grid, have a file annotation called `FileHeader` that states the start and end datetime range for the data in the file. For a file whose name is stored in the `fileName` string variable, the code for reading this datetime range is as follows:

```
import h5py
# read from file
with h5py.File( fileName, 'r' ) as fileHandle:
  FileHeader = fileHandle.attrs[ 'FileHeader' ].decode( 'utf-8' )
# -- convert from one long string to a dictionary of parameters
FileHeader = FileHeader.split( ';\n' )[0:-1]
FileHeader = dict( nameValue.split('=') for nameValue in FileHeader )
```

The contents of the `FileHeader` string includes these two parameters, shown here with a sample value:

```
StartGranuleDateTime: 2021-05-19T00:00:00.000Z
StopGranuleDateTime:  2021-05-19T00:29:59.999Z
```

IMERG HDF5 files, and no other GPM data products, contain a one-element `time` array that states the start datetime covered by the IMERG grid file. The value has units of seconds since the Unix epoch 1 January 1970 0000 UTC.

```
import h5py
import numpy
with h5py.File( fileName, 'r' ) as fileHandle:
  startSecondsSince19700101 = numpy.array( fileHandle['/Grid/time'] )[0]
```

As described in a subsequent section, the Goddard Earth Sciences (GES) DISC creates an IMERG-derived NetCDF4 file storing a 24-hour average precipitation rate. That NetCDF4 file contains a time array that is read somewhat differently because it is stored in the NetCDF4 format and because its units are days not seconds:

```
from netCDF4 import Dataset
with Dataset( fileName, 'r' ) as fileHandle:
  startDaysSince19700101 = numpy.array[ fileHandle['time'] )[0]
startSecondsSince19700101 = 24.0 * 60.0 * 60.0 * startDaysSince19700101
```

Once you have calculated seconds since the start of the Unix epoch, it is just one more step to creating a Python datetime object.

```
import datetime
startDatetime=datetime.datetime.utcfromtimestamp(startSecondsSince19700101)
```

5.2. Reading Datetime from Swath File

For a GPM swath file, each scan within the file has its own datetime. Depending on the instrument, each scan has a duration that may be somewhat less than a second or several seconds. The fields of view in a single scan form a line on the Earth's ellipsoid, almost always oriented approximately across the satellite's direction of motion. The datetime of each scan is stored in the variables in the `scantime` group of the HDF5 file. The datetime stored in GPM HDF5 files is stored in UTC time. In GPM swath files, the scan's datetime is stored in the variables `Year`, `Month`, `DayOfMonth`, `Hour`, `Minute`, `Second`, and `MilliSecond`. To read the seconds of the hour in the first swath of a 2A GPROF GMI HDF5 file, use the following Python commands:

```
import h5py
import numpy
with h5py.File( fileName, 'r' ) as fileHandle:
  secondsOfHour = numpy.array( fileHandle['/S1/ScanTime/Second'] )
```

6. Reading NetCDF4 and GeoTIFF Grid Files

The standard GPM data products have been written in the HDF5 format since the launch of the GPM satellite in 2014, but some derived products are written in the NetCDF4 or GeoTIFF format. In particular, NetCDF4 and GeoTIFF are used to store averages or accumulations that are derived from the IMERG HDF5 files. The *.zip file mentioned in Section 2 contains IMERG data stored as a NetCDF4 file and GeoTIFF. The `gpm.py` Python program contains the `readDAACdailyIMERGinNetCDF()` and `readIMERGtiffPPS()` functions for reading these two formats.

6.1. IMERG 24-hour Average Stored in a NetCDF4 File

As of 2022, Anaconda Python 3 does not include the NetCDF4 module. It can be installed in the following way: `conda install -c anaconda netcdf4`. After installation, use it in the following way:

```
from netCDF4 import Dataset
import numpy
grid = {}  # empty dictionary
with Dataset( fileName, 'r' ) as fileHandle:
  grid['data'] = numpy.array( fileHandle['precipitationCal'] )
  grid['lat' ] = numpy.array( fileHandle['lat'] )
  grid['lon' ] = numpy.array( fileHandle['lon'] )
  grid['time'] = numpy.array( fileHandle['time'] )
  grid['FileHeader'] = fileHandle.getncattr( 'FileHeader' )
grid.keys()
```

The one-element `time` array of the DAAC-created NetCDF4 file stores the start date of the observations in the file. The date has units of days since the Unix epoch of 1 January 1970 0000 UTC. To convert this date to a Python datetime object, use the following Python commands:

```
import datetime
startDaysSince19700101 = grid['time'][0]
startSecondsSince19700101 = 24.0 * 60.0 * 60.0 * startDaysSince19700101
startDatetime=datetime.datetime.utcfromtimestamp( \
  startSecondsSince19700101 )
```

By using the factor of 24×60×60 for seconds per day, the above calculation of `startSecondsSince19700101` assumes two things: no leap seconds occurred between 1 January 1970 and the present and the correct number of leap days are included in the value of `startDaysSince19700101`. Datetime calculations performed by common Linux, C, and Python libraries, such as the Python datetime library, generally make both of these assumptions. One can verify these facts for the case of the Python datetime function by seeing that the following Python commands produce an output of `2020-01-01 00:00:00.000000`:

```
import datetime
daysBetween1Jan1970and1Jan2020 = 18262.0
secondPerDay = 24.0 * 60.0 * 60.0
datetime1Jan2020 = datetime.datetime.utcfromtimestamp( \
  secondPerDay * daysBetween1Jan1970and1Jan2020 )
print( datetime1Jan2020.strftime('%Y-%m-%d %H:%M:%S.%f') )
```

While 27 leap seconds have occurred between the start of the Unix epoch and the start of 2020, the above code snippet demonstrates that the Python datetime library performs its calculations as if no leap seconds had occurred. Incidentally, the number of days (18,262) used in this example calculation was obtained from the IDL expression `julday(1,1,2020,0,0,0) - julday(1,1,1970,0,0,0)`.

Alternatively, one can obtain the start date of a GPM 24-hour IMERG NetCDF4 file by examining the `grid['FileHeader']` string that Python reads from the file's `FileHeader` file annotation.

6.2. IMERG Accumulation Stored in a GeoTIFF and WorldFile

The detailed documentation of the IMERG accumulations stored in GeoTIFF files is available in the IMERG GIS data product documentation, which can be downloaded form this URL: https://arthurhou.pps.eosdis.nasa.gov/Documents/README.GIS.pdf. That document describes how to read the IMERG GIS product in Python and in other languages. The following paragraphs provide an abbreviated explanation focused on Python.

The simplest way to read a GeoTIFF in Python is to read it as if it were a simple TIFF file and then read the separate WorldFile to obtain the needed geographic metadata. The Python PIL library can read data from a TIFF file. The `gpm.py` program uses this approach, which in brief is the following Python code:

```
from PIL import Image
fileHandle = Image.open( fileName )
grid = {} #create empty dictionary
grid['data'] = numpy.array( fileHandle )
grid['data'].shape
```

PPS stores geographic metadata inside the GeoTIFF file, but PPS also stores the same metadata in a simpler-to-read format outside of the GeoTIFF file. Specifically, for each GeoTIFF file that PPS creates, PPS also creates an ESRI WorldFile. A WorldFile is a small text file that contains geographic metadata (https://en.wikipedia.org/wiki/World_file). The gpm.py Python program contains a function called `readWorldFile()` for reading WorldFiles. For a GeoTIFF with file extension of `*.tif`, the associated WorldFile has an identical name except for a file extension of `*.tfw`. The following lines of Python code calculate the latitude range, longitude range, and resolution of a grid based on the contents of a WorldFile and the shape of the numpy array read in the code snippet above using the PIL library:

```
shape = grid['data'].shape
# -- read worldfile and make contents a Python array
with open( worldFileName ) as fileHandle:
  record = fileHandle.readlines()
worldFile = [ float(oneRecord) for oneRecord in record ]
# -- calculate the range and resolution of the grid
dLon = abs( worldFile[0] )
dLat = abs( worldFile[3] )
minLon = worldFile[4] - dLon/2
maxLat = worldFile[5] + dLat/2
numLat = shape[0]
numLon = shape[1]
minLat = round( maxLat - dLat*numLat, 2 )
maxLon = round( minLon + dLon*numLon, 2 )
maxLat = round( maxLat, 2 )
minLon = round( minLon, 2 )
print( minLat, maxLat, minLon, maxLon )
```

A third option for obtaining geographic metadata is to read the file specification document for the IMERG HDF5 files and IMERG GIS files, both available on the PPS website, https://arthurhou.pps.eosdis.nasa.gov/.

References and Websites

Anaconda: Python distribution. https://anaconda.org/.

Collette, A., 2013: *Python and HDF5*. O'Reilly Media, Inc. [Available in hardcopy or through http://safaribooksonline.com]

Data Product Development Group (DPDG) Working Group, July 2020: *Data Product Development Guide for Data Producers*. NASA, report ESDS-RFC-041, https://cdn.earthdata.nasa.gov/conduit/upload/14909/ESDS-RFC-041.pdf.

Hou, A. Y., R. K. Kakar, S. Neeck, A. A. Azarbarzin, C. D. Kummerow, M. Kojima, R. Oki, K. Nakamura, and T. Iguchi, 2014: The Global Precipitation Measuring Mission. *Bulletin Am. Meteorological Society*, May 2014, 701–722, doi: https://doi.org/10.1175/BAMS-D-13-00164.1.

Huffman, G. J., D. T. Bolvin, D. Braithwaite, K. Hsu, R. Joyce, C. Kidd, E. J. Nelkin, S. Sorooshian, J. Tan, and P. Xie, 2019: Algorithm Theoretical Basis Document (ATBD) Version 06 NASA Global Precipitation Measurement (GPM) Integrated Multi-satellitE Retrievals for GPM (IMERG). https://gpm.nasa.gov/sites/default/files/document_files/IMERG_ATBD_V06.pdf.

Lutz, M, 2020: Python Changes 2014+. web page, https://learning-python.com/python-changes-2014-plus.html.

Lutz, M., 2013: *Learning Python*, 5th edition. O'Reilly Media, Inc. [Available in hardcopy or through http://safaribooksonline.com]

NASA: Global Precipitation Measurement. website, http://pmm.nasa.gov.

PPS, 2021: *File Specification for GPM Products*. NASA, https://gpmweb2https.pps.eosdis.nasa.gov/pub/GPMfilespec/filespec.GPM.pdf.

PPS: Precipitation Processing System (PPS). homepage, NASA, https://arthurhou.pps.eosdis.nasa.gov/.

PPS: STORM data ordering system. web site, NASA, http://storm.pps.eosdis.nasa.gov.

Python array library, http://www.numpy.org.

Python HDF5 library, http://www.h5py.org.